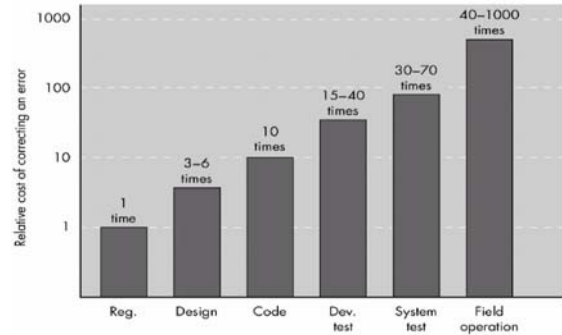


Software Quality & Metrics

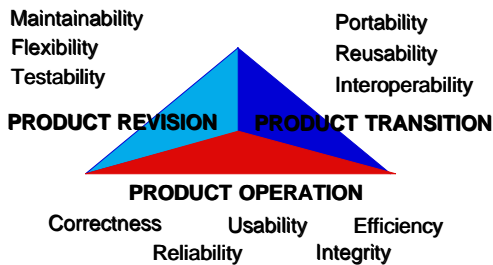
Sources:

1. Roger S. Pressman, *Software Engineering – A Practitioner’s Approach*, 5th Edition, ISBN 0-07-365578-3, McGraw-Hill, 2001 (Chapters 8 & 19)
2. Measurement for Software Process Improvement by Barbara Kitchenham (Chapter 4) (out of print)

Why SQA Activities Pay Off?



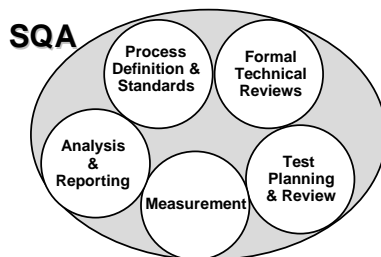
McCall’s Triangle of Quality



Quality Concepts

- general objective: reduce the “variation between samples” ... but how does this apply to software?
- quality control: a series of inspections, reviews, tests
- quality assurance: analysis, auditing and reporting activities
- cost of quality
 - appraisal costs
 - failure costs
 - external failure costs

Software Quality Assurance



Software Quality

Quality is a complex and multifaceted concept:

1. The *transcendental view* of philosophy which regards quality as something that can be recognised but not defined;
2. The *user view*, which is usually encapsulated in the phrase “fitness for purpose”;
3. The *manufacturing view*, which is usually encapsulated in the phrase “conformance to specification”;
4. The *product view*, which relates quality to the inherent characteristics of the product;
5. The *value for money view*, which considers quality to be conditional on the amount a customer is willing to pay.

User View

- The **user view** regards a quality product as one with the characteristics (functions and attributes) that meet its user's needs.
- The user view of quality is based on an evaluation of the product **in the context** of the task it is intended to perform.
- It is the viewpoint that is inherent in reliability and performance modelling which are based on the concept of assessing product behaviour with respect to **operational profiles**.

Manufacturing View

- This is the approach implicit in process standards such as ISO 9001 and the Capability Maturity Model.
- Advocates of this approach usually concentrate on conformance to process rather than conformance to specification.
- However, in the world of software, there is no guarantee that either conformance to process standards such as ISO 9001, or achieving high maturity levels will deliver good products.
- The process standards are based on the principle of "Documenting what you do and doing what you say".

Product View

- The **product view** considers the inherent characteristics of the product.
- This view is frequently adopted by advocates of software metrics who assume that measuring and controlling internal product properties (i.e. internal quality indicators) will result in improved external product behaviour (i.e. quality in use).
- Assessing quality by measuring internal properties is attractive because it offers an objective and context independent view of quality.
- Software engineering researchers have developed models that attempt to link the product view to the user view.

User View Models

- One important class of measures taking the user view of software are related to **software reliability** measurement.
- Software reliability concerns the frequency with which a software product fails when it is being used - a concept of understandable importance to a user.
- The relationship between reliability model and the user view of quality depends on either being able to observe the behaviour of software in the field, or simulate user behaviour by means of "**operational profiles**".
- "Operational profiles" are also important for assessing software and **system performance** where we are interested in measuring characteristics such as job throughput, response times, central processing unit occupancy etc.
- For usability, we need to think in terms of "task profiles" rather than "operational profiles". Usability assessment can be measured in terms of times to perform tasks, and task error rates.

Product View Models

Quality Characteristic	Definition
Functionality	A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.
Reliability	A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.
Usability	A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.
Efficiency	A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
Maintainability	A set of attributes that bear on the effort needed to make specified modifications. Note. Modifications may include corrections, improvements or adaptations of software to changes in the environment and in the requirements and functional specifications.
Portability	A set of attributes that bear on the ability of software to be transferred from one environment to another. Note. The environment may include organisational, hardware or software environment.

Procedure

- Identify for each component which quality-carrying properties must be satisfied and which high-level quality attribute each property affects
- From the top down, you need to consider for each quality attribute which quality-carrying properties imply that attribute and which components possess those properties

Categories of Properties

1. correctness properties (minimal generic requirements for correctness);
2. structural properties (low-level, intramodule design issues);
3. modularity properties (high-level, intermodule design issues);
4. descriptive properties (various forms of specification/documentation).

Correctness Properties

Id	Property	Definition	Structural form	Quality impact	Sample defects
C1	Computable	Result obeys standard theory of computation	Expressions	Reliability Functionality	Subscript out of range Divide by zero
C2	Complete	All elements of structural form satisfied	Objects Modules Statements	Functionality Reliability Usability Maintainability	Self assignment Module generating no output Unreachable code in a selection mechanism
C3	Assigned	Variable given value before use.	Variables	Reliability Functionality	Use of undefined variable on right hand side of expression
C4	Precise	Adequate accuracy is preserved in calculations	Variables Constants	Reliability Functionality	Use of single precision when double is required

Structural Properties

Id	Property	Definition	Structural form	Quality impact	Sample defects
S1	Structured	Single entry and exit	Sequences Guards Expressions	Functionality Reliability Maintainability	Exit from middle of loop Multiple returns from function
S2	Resolved	Matched data and control structure	Sequences Guards Expressions	Maintainability Efficiency	Use of a single loop to process two-dimensional array
S3	Homogeneous	Only conjunctive invariants for loops	Loops Modules (recursive)	Maintainability	Loop structure with non-cohesive functionality
S4	Effective	No computational redundancy	Expression Statements	Usability Efficiency Maintainability	Expression with unnecessary computation

Modularity Properties

Id	Property	Definition	Structural form	Quality impact	Sample defects
M1	Parameterized	All inputs accessed via a parameter list	Modules	Maintainability Reusability Portability	Module with no parameters
M2	Loosely coupled	Data coupled	Module calls	Reliability Maintainability Reusability Portability	Modules exhibiting other forms of coupling (e.g. links based on use of common data).
M3	Encapsulated	Uses no global variables	Variables Constants Types	Reliability Maintainability Reusability Portability	Use of variable in a module that has not been declared within the module's scope
M4	Cohesive	Relationships between elements of an entity are maximized	Sequences	Maintainability Reusability Portability	Loop with dispersed initialisation

Descriptive Properties

Id	Property	Definition	Structural form	Quality impact	Sample defects
D1	Specified	Preconditions and Post conditions provided	Objects Modules Loops Sequences	Functionality Reliability Maintainability Reusability Portability Usability	Specification is ambiguous, inaccurate or inconsistent No pre- or post-conditions
D2	Documented	Comments associated with all blocks	Objects Modules Loops Sequences Module calls Data structures Variables Constants Types	Maintainability Reusability Portability Usability	More comments than required Documentation misleading or wrong
D3	Self-descriptive	Identifiers have meaningful names	Objects Modules Module calls Data structures Variables Constants	Maintainability Reusability Portability Usability	Name is misleading, ambiguous or wrong Names bear no relation to function

Manufacturing View – Defect Counts

- Defect counts are the number of known defects recorded against the product during its development and live use
- For comparison purposes it is important that defect counting is always applied to the same stages in the lifecycle.
- For more detailed analysis it is useful to categorise defects on the basis of the phase/activity that introduced the defect and the phase/activity in which the defect was detected.
- To measure product quality using defect counts, it is important to note that pre-release defect rates are a surrogate measure of quality.
- Post-release defect rates are the real measure of the quality of the delivered product.

Manufacturing View – Defect Counts II

- It is also important to ensure that defects are recorded at comparable stages post-release (e.g. up to six months after release).
- In order to compare the quality of different products, defect counts should be "normalised" by dividing by product size to give defect rates.
- In addition, post-release defect counts should be normalised with respect to the number of different product users and/or number of different computer installations which use the software product, and the amount of use by each installation/user

Manufacturing View – Rework Effort

- In software development, rework costs are usually equated to staff effort spent correcting defects before and after release.
- It is important to define what is meant by rework during software development.
- It is best to exclude end-phase verification and validation, and include only rework of documents and code that have been formally signed-off.
- Debugging effort expended during integration and system testing should be included.
- To compare different products, rework effort is normalised by being calculated as a percentage of development effort.

Manufacturing View – Rework Effort II

- Only defect correction should count as re-work because it is a cost of non-conformance. The ability to enhance software to include new facilities is not a cost, it is usually a benefit!
- It is important to separate pre- and post-release rework effort. Post-release rework effort is a measure of *delivered quality*. Pre-release rework effort is a measure of *manufacturing efficiency*.
- It is also a cost of non-conformance, but if the effort can be attributed to specific phases/activities it can also be used to identify areas of maximum leverage for process improvement.

Software Quality Metrics

- Conformance to software requirements is the foundation from which quality is measured.
- Specified standards define a set of development criteria that guide the manner in which software is engineered.
- Software quality is suspect when a software product conforms to its explicitly stated requirements and fails to conform to the customer's implicit requirements (e.g. ease of use).

Metrics for specification Quality

- Davis and his colleagues propose a list of characteristics that can be used to assess the quality of the analysis model and the corresponding requirements specification: *specificity* (lack of ambiguity), *completeness*, *correctness*, *understandability*, *verifiability*, *internal and external consistency*, *achievability*, *concision*, *traceability*, *modifiability*, *precision*, and *reusability*.
- Although many of these characteristics appear to be qualitative in nature, Davis et al. suggest that each can be represented using one or more metrics. For example, we assume that there are n_r requirements in a specification, such that

$$n_r = n_f + n_{nf}$$

where n_f is the number of functional requirements and n_{nf} is the number of non-functional (e.g., performance) requirements.

The *specificity* (lack of ambiguity) of requirements can be determined:

$$Q_1 = n_{ui}/n_r$$

where n_{ui} is the number of requirements for which all reviewers had identical interpretations. The closer the value of Q to 1, the lower is the ambiguity of the specification.

The *completeness* of functional requirements can be determined by computing the ratio:

$$Q_2 = n_u/[n_i \times n_s]$$

where n_u is the number of unique function requirements, n_i is the number of inputs (stimuli) defined or implied by the specification, and n_s is the number of states specified. The Q_2 ratio measures the percentage of necessary functions that have been specified for a system. However, it does not address nonfunctional requirements. To incorporate these into an overall metric for completeness, we must consider the degree to which requirements have been validated:

$$Q_3 = n_v/[n_c + n_{nv}]$$

where n_v is the number of requirements that have been validated as correct and n_{nv} is the number of requirements that have not yet been validated.

Architectural Design Metrics

- *Architectural design metrics*
 - Structural complexity = g(fan-out)
 - Data complexity = f(input & output variables, fan-out)
 - System complexity = h(structural & data complexity)
- *HK metric*: architectural complexity as a function of fan-in and fan-out
- *Morphology metrics*: a function of the number of modules and the number of interfaces between modules

Component-Level Design Metrics

- *Cohesion metrics*: a function of data objects and the locus of their definition
- *Coupling metrics*: a function of input and output parameters, global variables, and modules called
- *Complexity metrics*: hundreds have been proposed (e.g., **Cyclomatic complexity**) We will now look at the cyclomatic complexity

McCabe's Cyclomatic Number

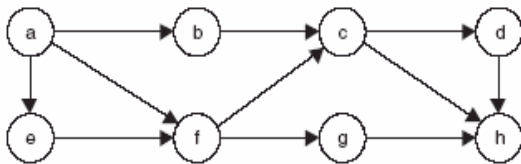
- The premise is that complexity is related to the control flow of program.
- Graph theory uses a formula, $C=e-n+1$ to calculate cyclomatic number. McCabe uses the slightly modified formula: $C=e-n+2p$, where:

e = Number of edges

n = Number of nodes

p = Number of strongly connected components (which is normally 1)

Example 1: Determine the cyclomatic number from the control flow graph shown below:

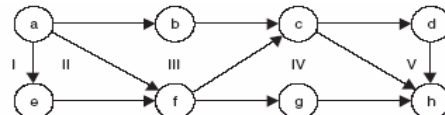


There are 8 nodes, so $n = 8$. There are 11 arcs, so $e = 11$. The cyclomatic number is $C = 11 - 8 + 2 = 5$

Planar Graph

- A planar graph is a graph that can be drawn without lines crossing.
- Euler L (1707-1783) proved that for planar graph that $2 = n - e + r$, where r = number of regions, e = number of edges, and n = number of nodes.
- A region is an area enclosed (or defined) by arcs, $r = e - n + 2$.
- Therefore, the number of regions on a planar graph equals the cyclomatic number.

Example: Label the regions in the control flow graph above -

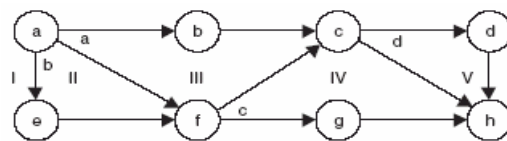


As shown, there are 5 regions. Region I is the outside of the graph

- Calculating the cyclomatic number from control graph is time-consuming.
- Constructing a control graph from a large program would be prohibitively time-consuming.
- McCabe found that the number of region is usually equal to one more than the number of decisions in a program, $C = \Pi + 1$, where Π is the number of decisions.
- In source code, an IF, a WHILE loop, or a FOR loop is considered one decision. A CASE statement or other multiple branch is counted as one less decision than the number of possible branches.
- Control flow graphs are required to have a distinct starting node and a distinct stopping node. If this is violated, the number of decisions will not be one less than the number of regions

Example2 : Label the decisions in the control flow graph of example 1 above with lower case letters.

As shown below, from node a, there are three arcs, so there must be two decisions, labeled a and b. From nodes c and f, there are two arcs and so one decision each. The other nodes have at most one exit and so no decisions. There are four decisions, so $C = 4 + 1 = 5$



Quality assurance and standards

- Standards are the key to effective quality management.
- They may be international, national, organizational or project standards.
- **Product standards** define characteristics that all components should exhibit e.g. a common programming style.
- **Process standards** define how the software process should be enacted.

Importance of standards

- Encapsulation of best practice- avoids repetition of past mistakes.
- They are a framework for quality assurance processes - they involve checking compliance to standards.
- They provide continuity - new staff can understand the organisation by understanding the standards that are used.

Product and process standards

Product standards	Process standards
Design review form	Design review conduct
Requirements document structure	Submission of documents to CM
Method header format	Version release process
Java programming style	Project plan approval process
Project plan format	Change control process
Change request form	Test recording process

Problems with standards

- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious manual work is often involved to maintain the documentation associated with the standards.

Standards development

- Involve practitioners in development. Engineers should understand the rationale underlying a standard.
- Review standards and their usage regularly. Standards can quickly become outdated and this reduces their credibility amongst practitioners.
- Detailed standards should have associated tool support. Excessive clerical work is the most significant complaint against standards.

Summary

- Quality is a complex concept that means different things to different individuals. It can be highly context dependent
- No simple measure of quality that will be accepted by everyone
- Define what aspect of quality you are interested in, and how you will measure it
- Define quality in a measurable way helps other people understand your viewpoint